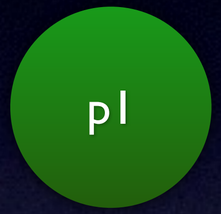
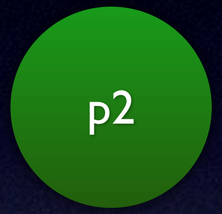
Операционные системы

лекция 6

hexlet.org

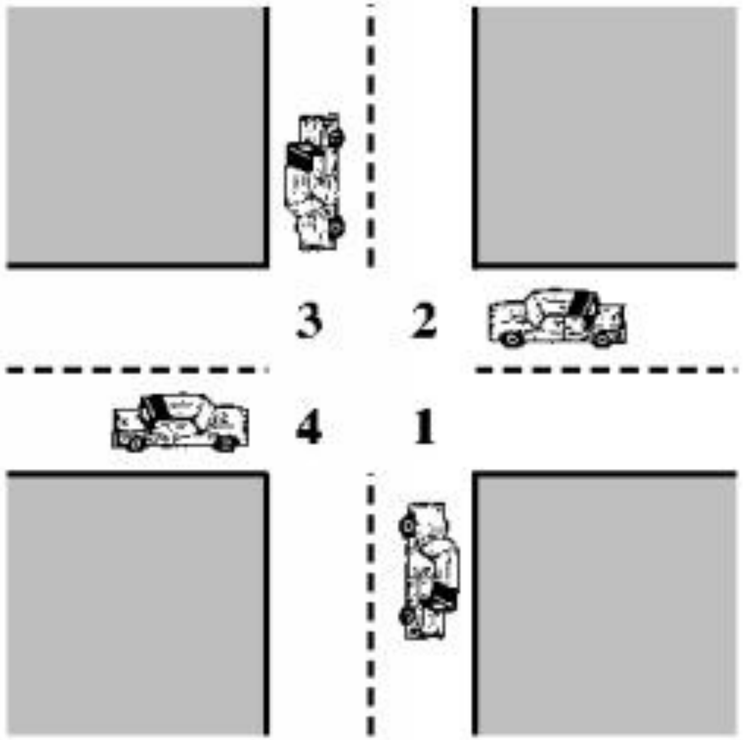
deadlock



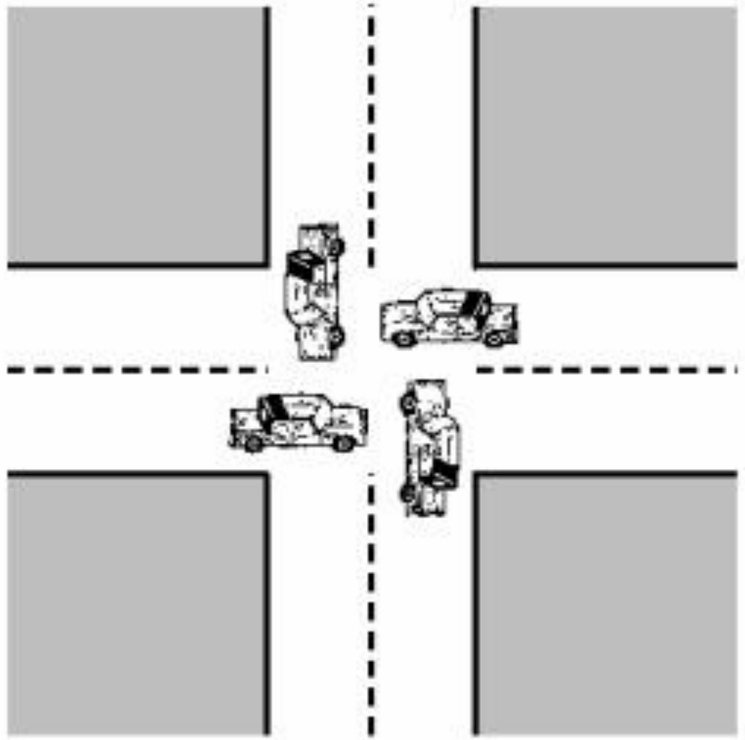



```
lock A
…
lock B
…
…
…
unlock B
unlock A
```

```
lock B
…
lock A
…
…
…
unlock A
unlock B
```



(a) Deadlock possible



(b) Deadlock


BANCO REAL
11455

Категории ресурсов

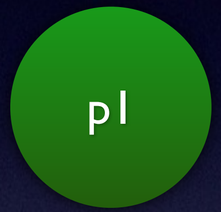
- Повторного использования. Используется одним процессом в один момент времени и не заканчивается.
- Расходуемые. Создается и потребляется.

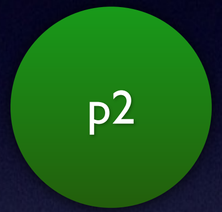
Reusable
(ресурсы повторного использования)

- ЦП, память, файлы, базы данных, семафоры
- Взаимная блокировка возможна если процесс использует один ресурс и запрашивает новый.

Consumable (расходуемые ресурсы)

- Прерывания, сигналы, сообщения
- Взаимная блокировка возможна если процесс ожидания сигнала – блокирующий






```
ожидание сообщения от p2
отправка сообщения в p2
```

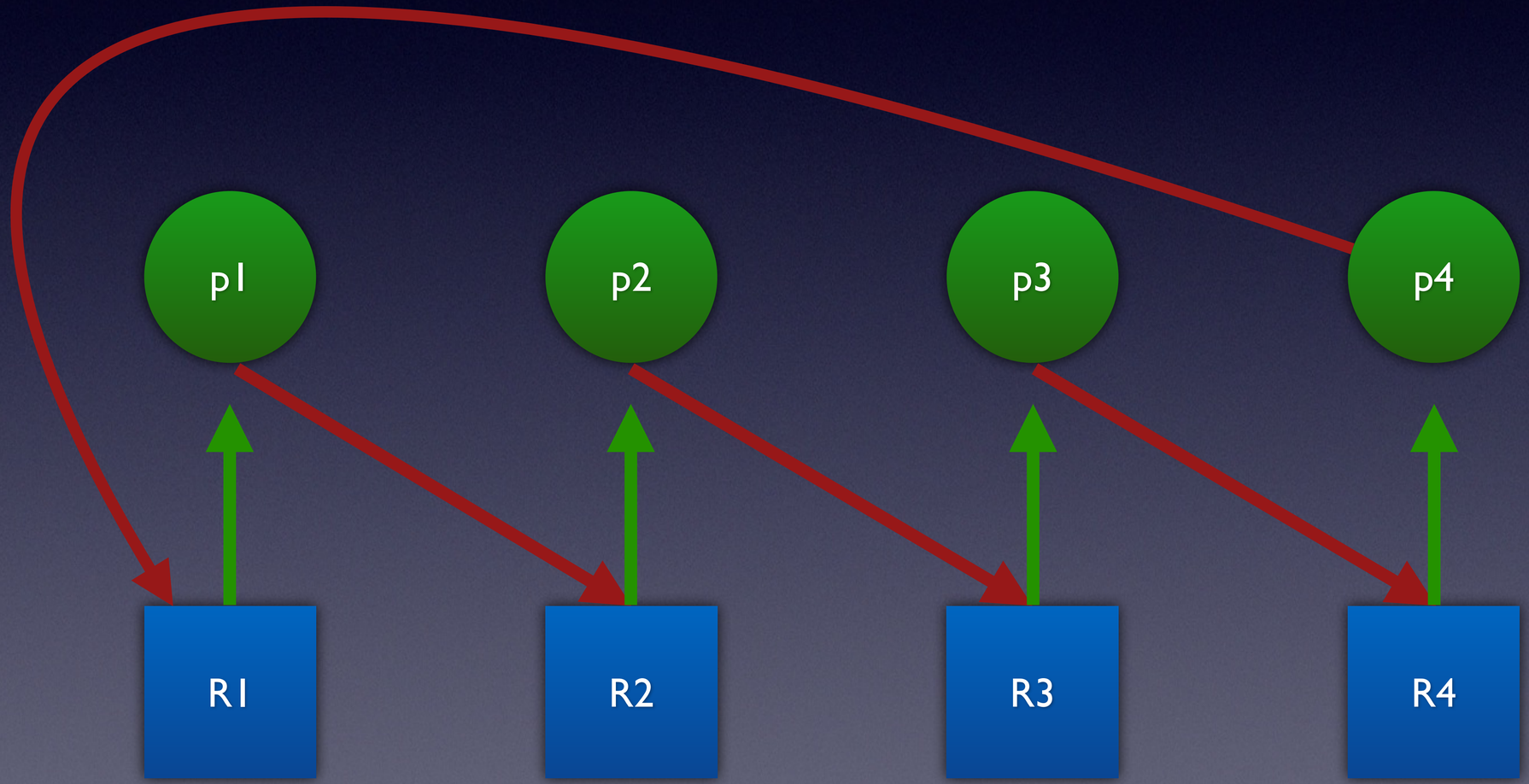
```
ожидание сообщения от p1
отправка сообщения в p1
```

Условия возникновения дедлоков

1. Взаимное исключение
2. hold and wait (процесс блокирует ресурс и ждет освобождения другого ресурса)
3. Система не может “отбирать” ресурсы у процесса
4. Циклическое ожидание (p2 ждет p2, p2 ждет p3, p3 ждет p1)

все условия должны быть соблюдены

Циклическое ожидание



Три аспекта взаимной блокировки

- Предотвращение
- Избегание
- Обнаружение

Предотвращение

- Косвенное: не позволять трем условиям выполняться одновременно
  - Взаимное исключение (должно поддерживаться на уровне ОС)
  - Hold and wait (процесс обязан запрашивать все необходимые ресурсы одновременно)
  - “Отбор” ресурсов системой (процесс должен освободить ресурс и запросить его снова)
- Прямое: запрещать циклическое ожидание

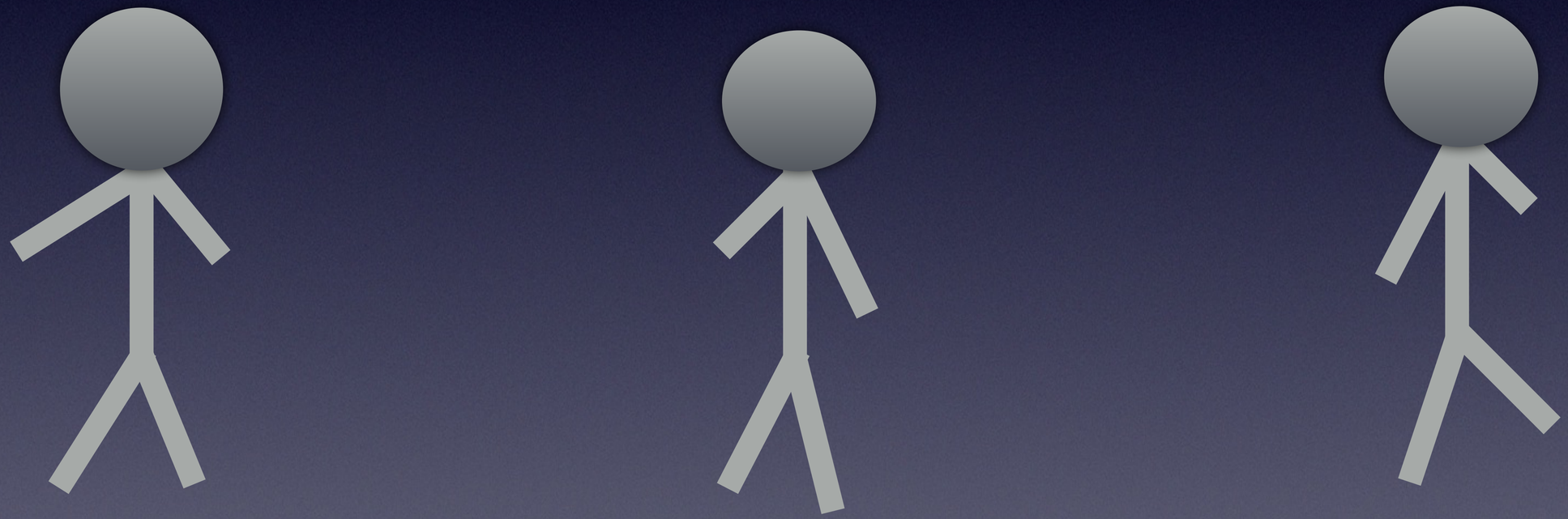
Избегание

- Оценка текущей ситуации – может ли она привести к взаимной блокировке в будущем?
- Необходимо знать будущие запросы процессов

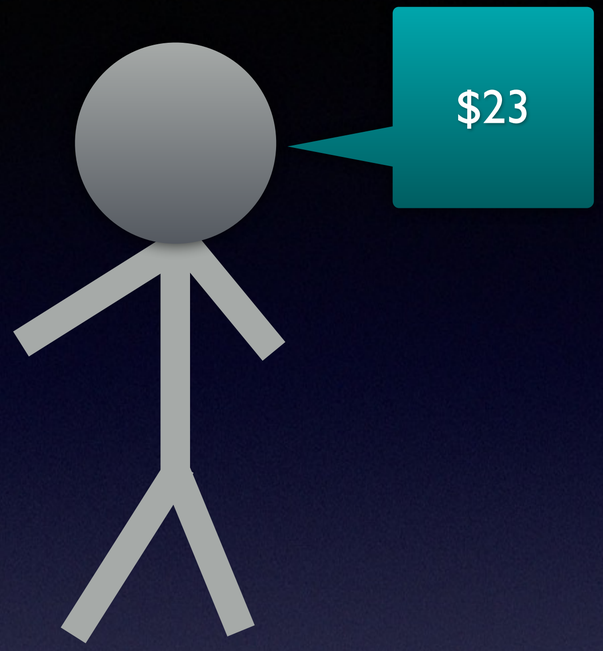
Два подхода к избеганию

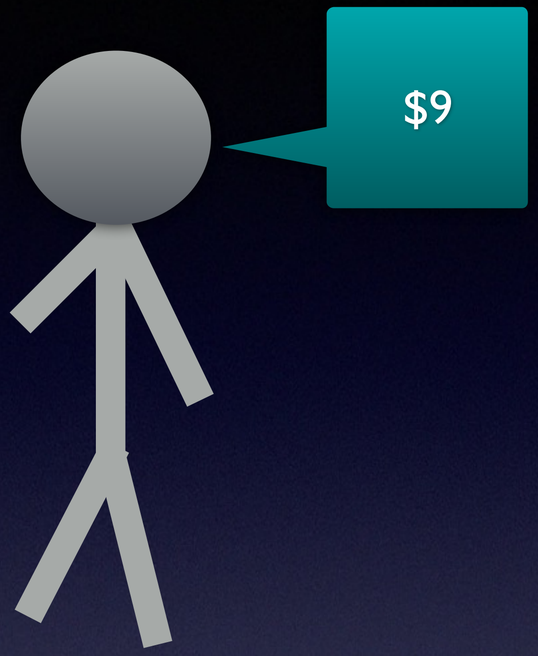
- **Запрет инициализации процесса**, который может привести к взаимной блокировке
  - не оптимально, рассчитано на худший сценарий
- **Запрет предоставления ресурсов** если это может привести к взаимной блокировке
  - рассмотрим Алгоритм банкира

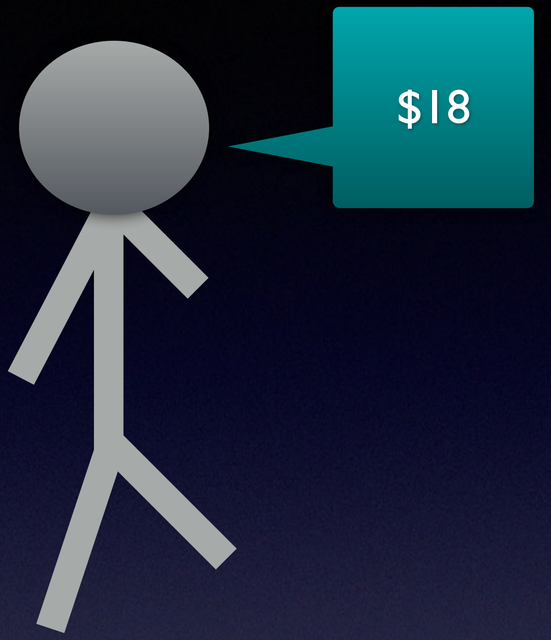
Алгоритм банкира



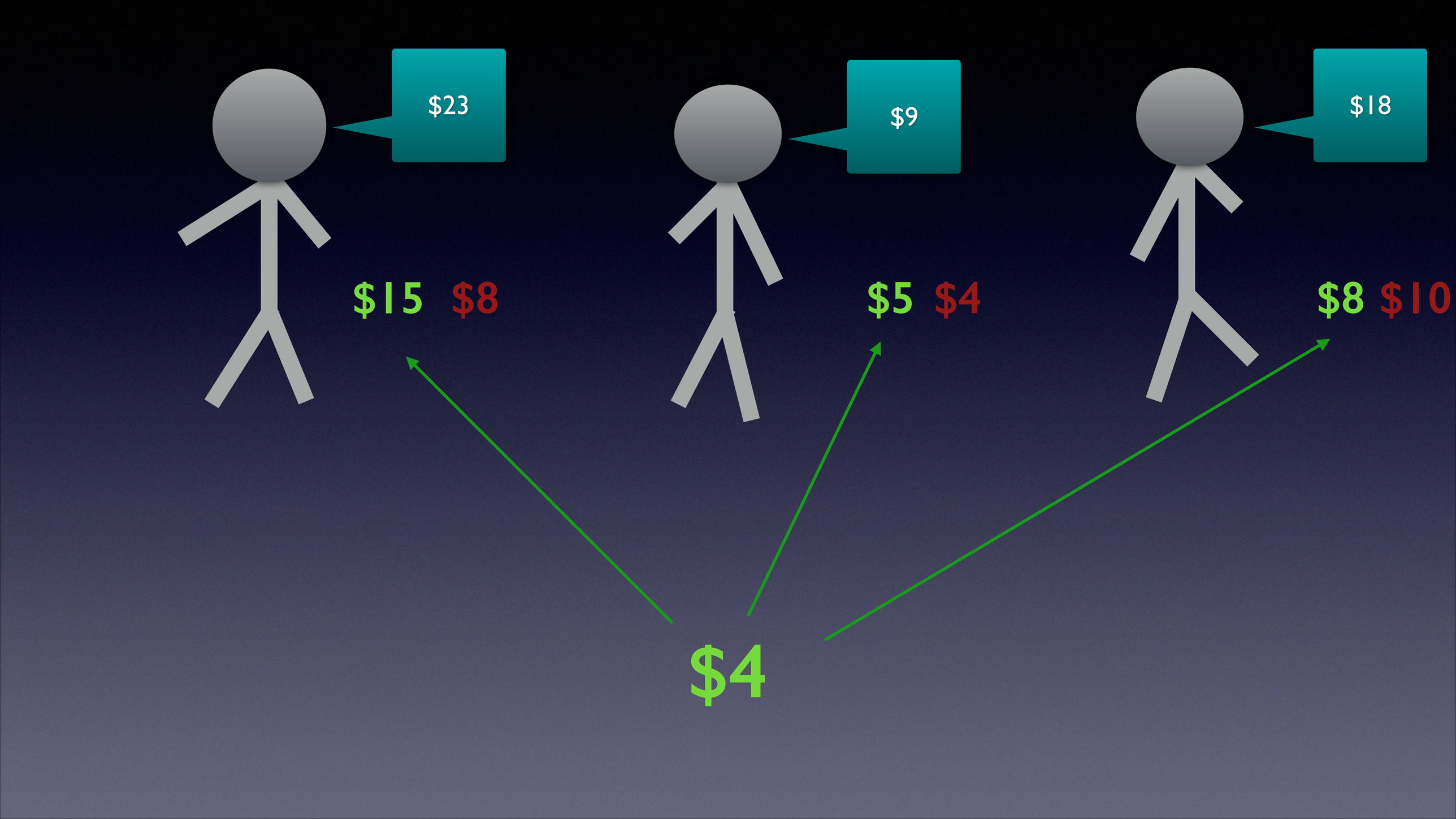
$32

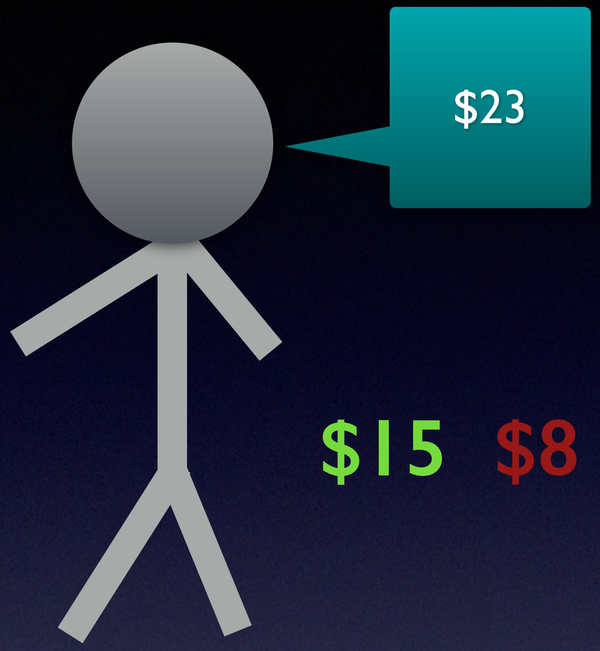




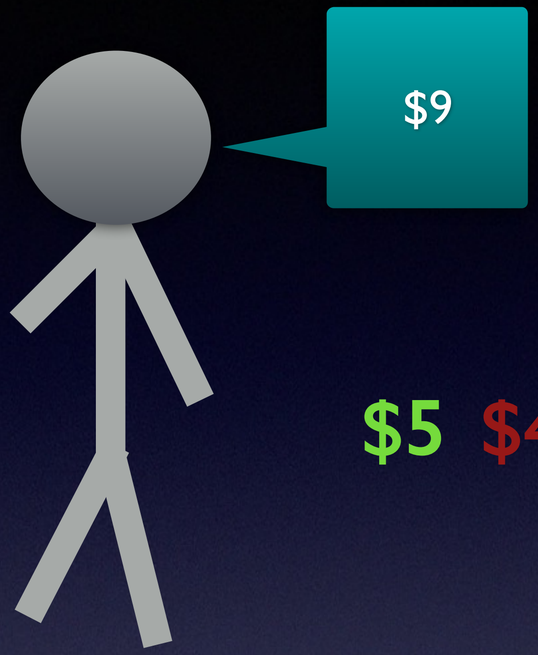


$32

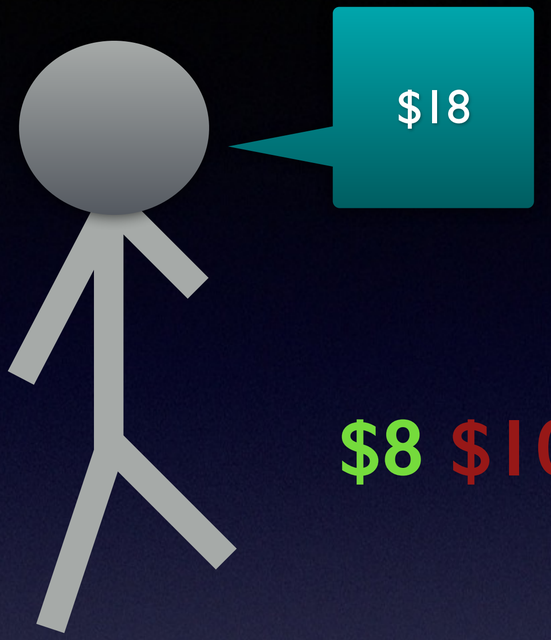

$23
$9
$18
$15 $8
$5 $4
$8 $10
$4



$15 $8



$5 $4

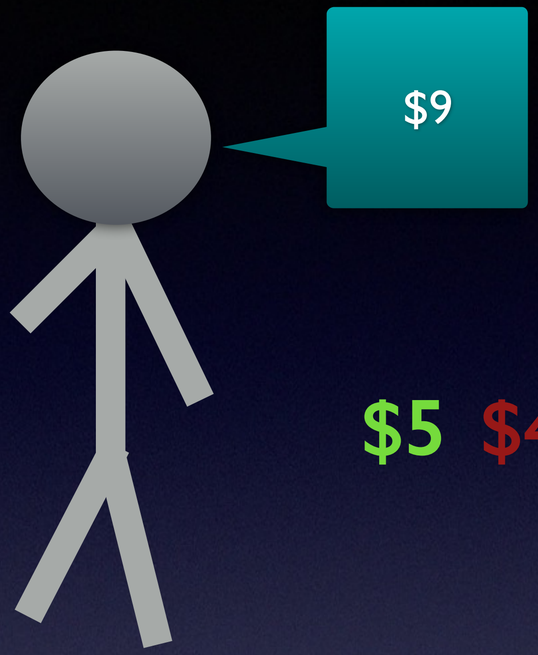


$8 $10

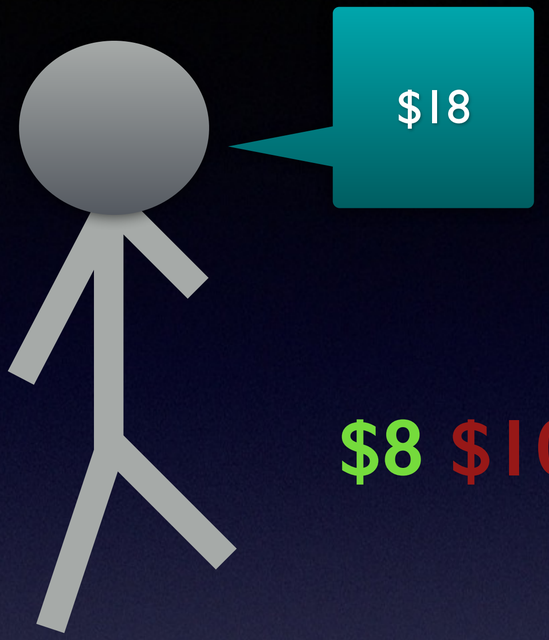
безопасное состояние

$4

$16 $7



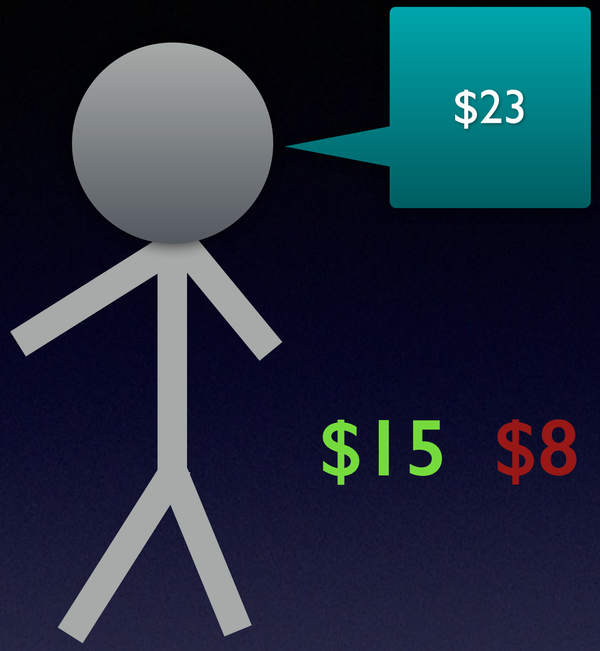
$5 $4



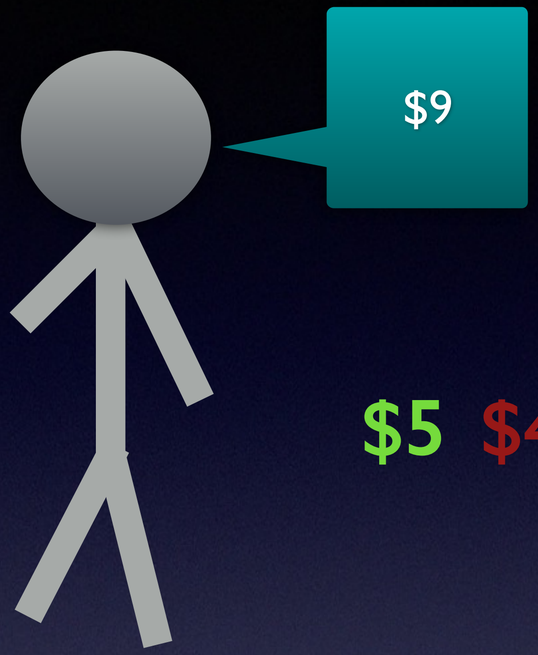
$8 $10

небезопасное состояние

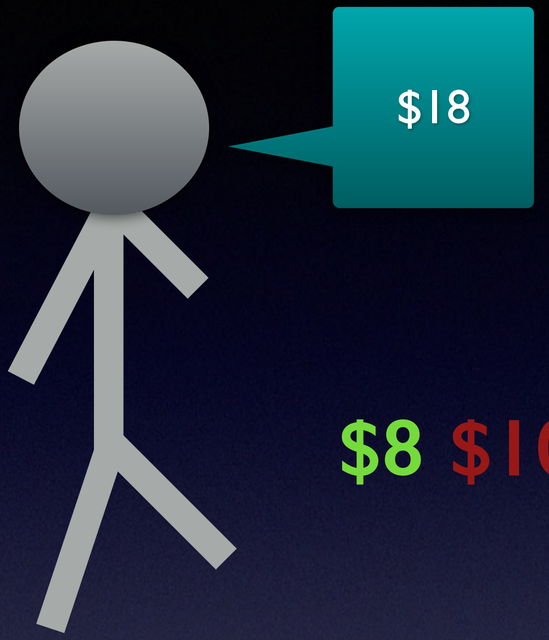
$3



$15 $8



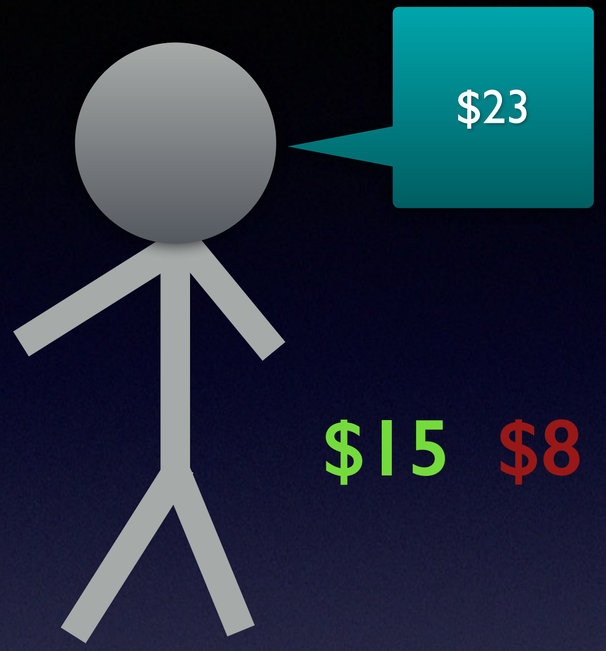
$5 $4



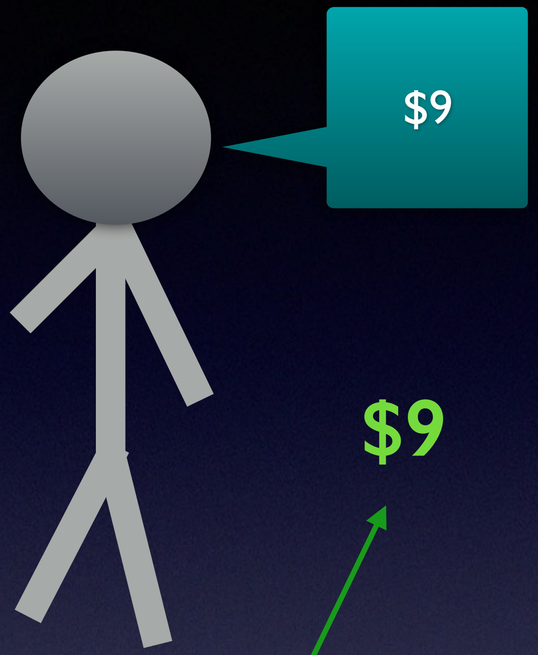
$8 $10

безопасное состояние

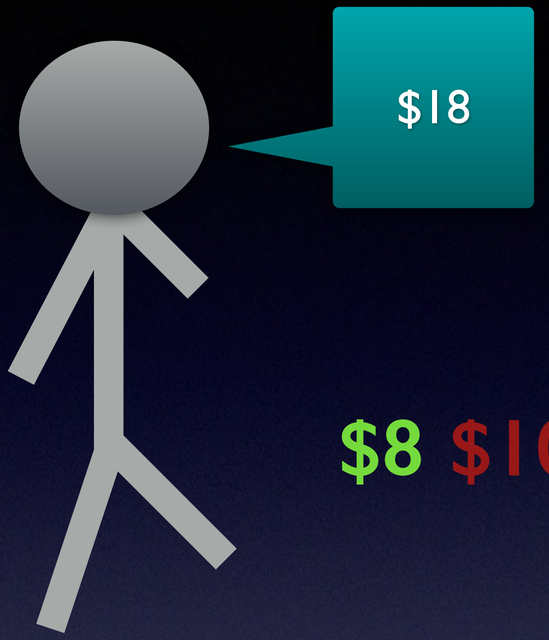
$4



$15 $8

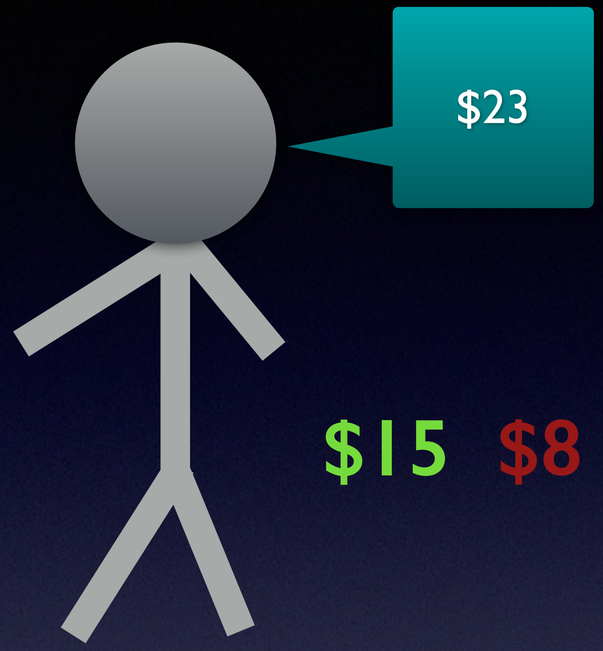


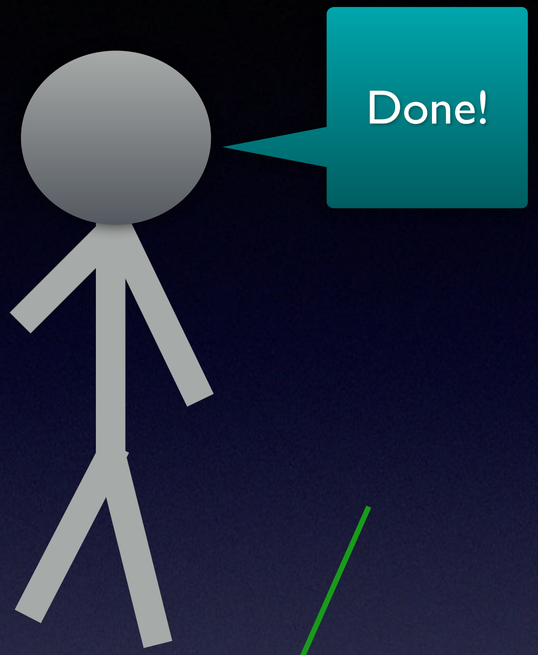
$9

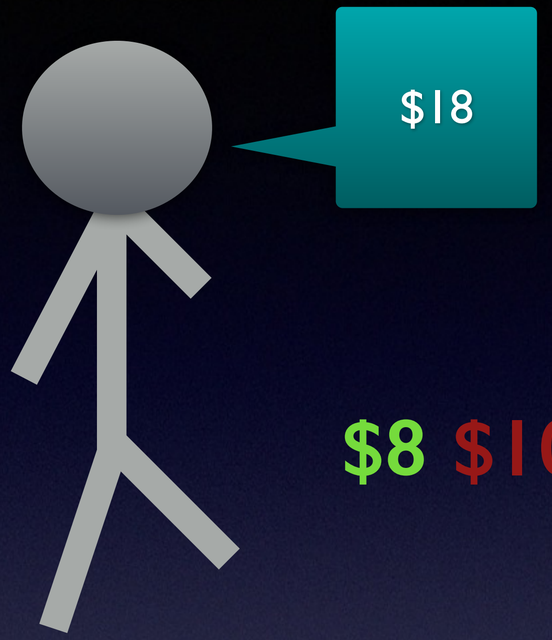


$8 $10

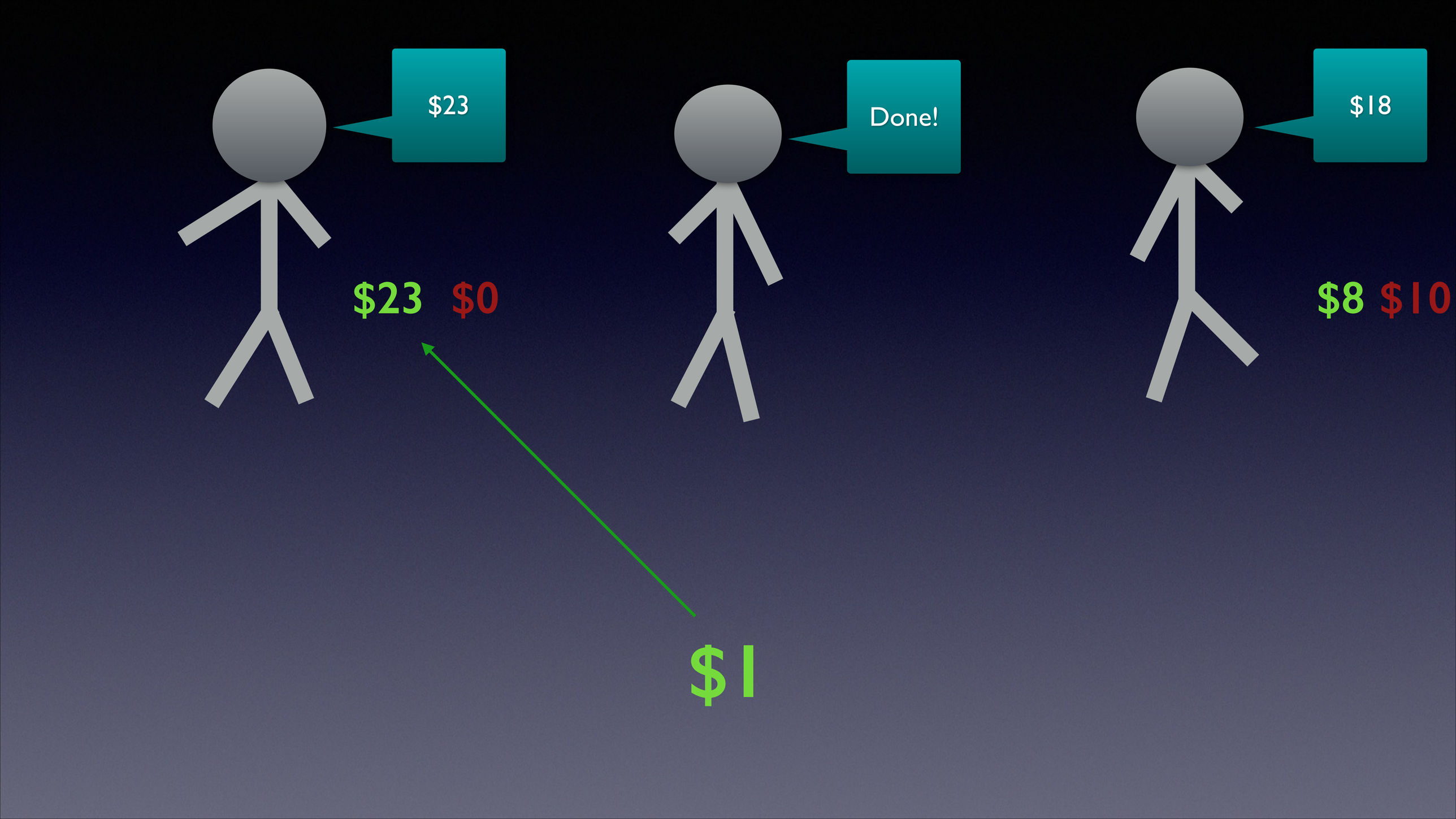
$0

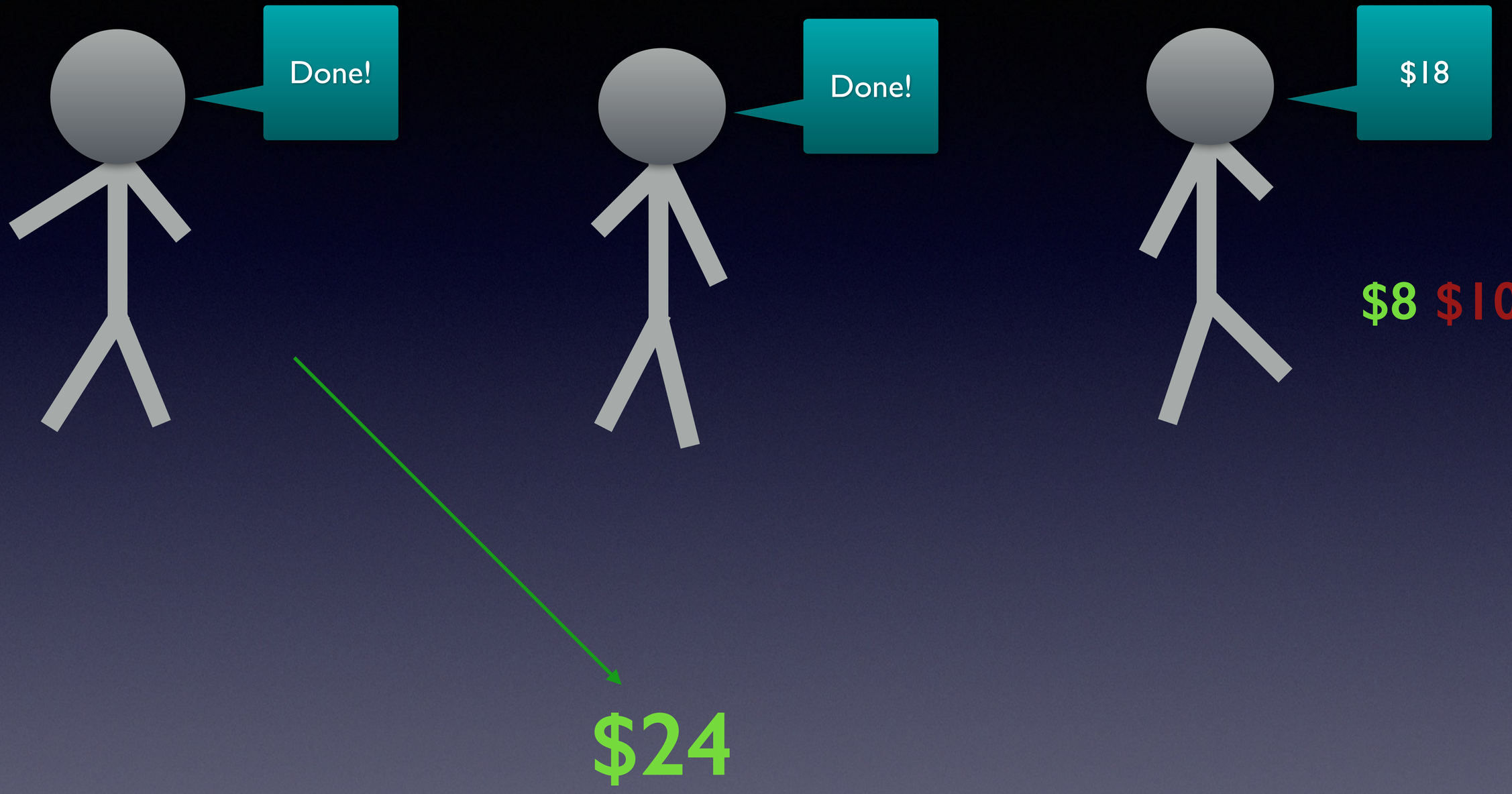


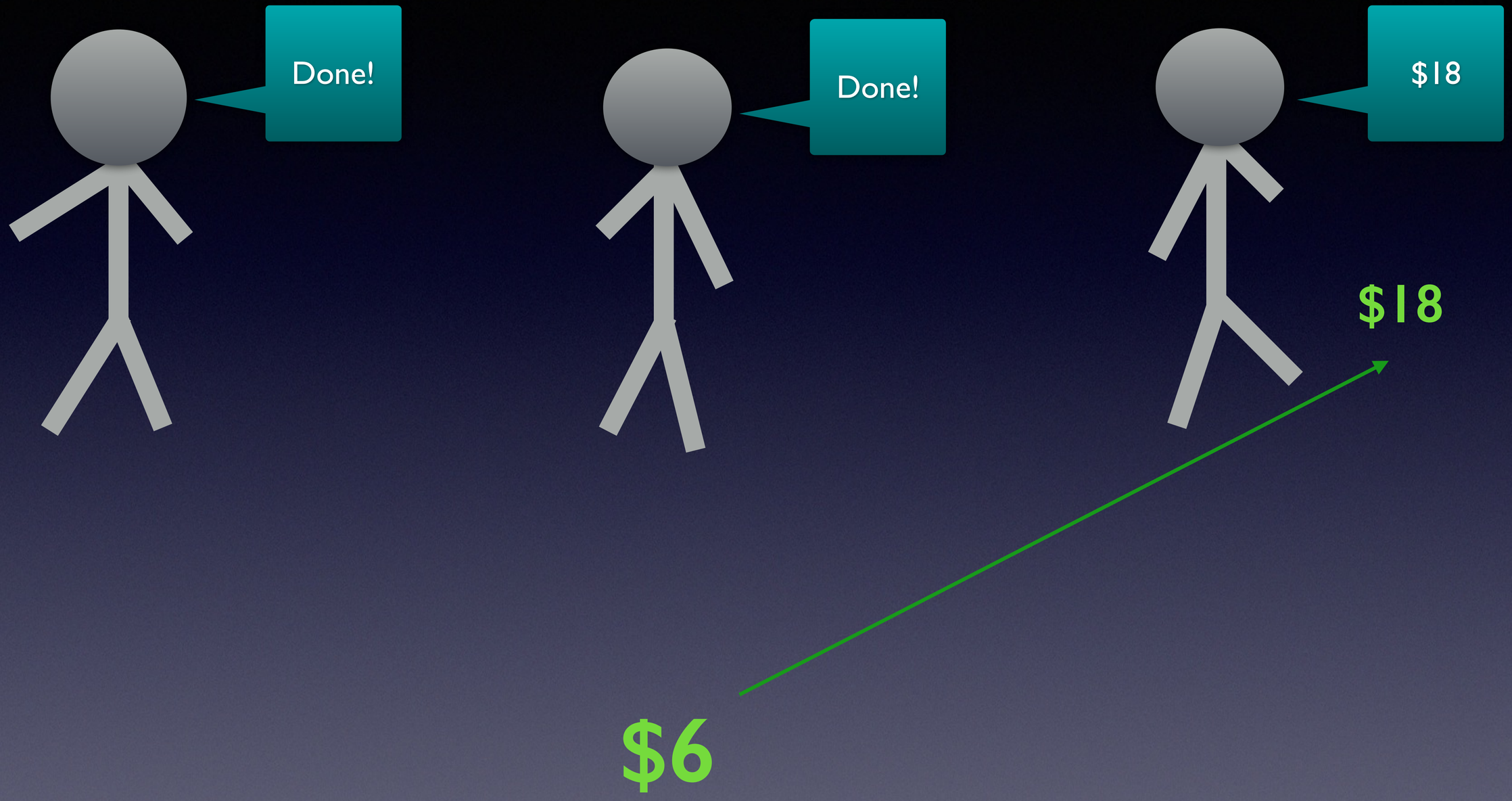


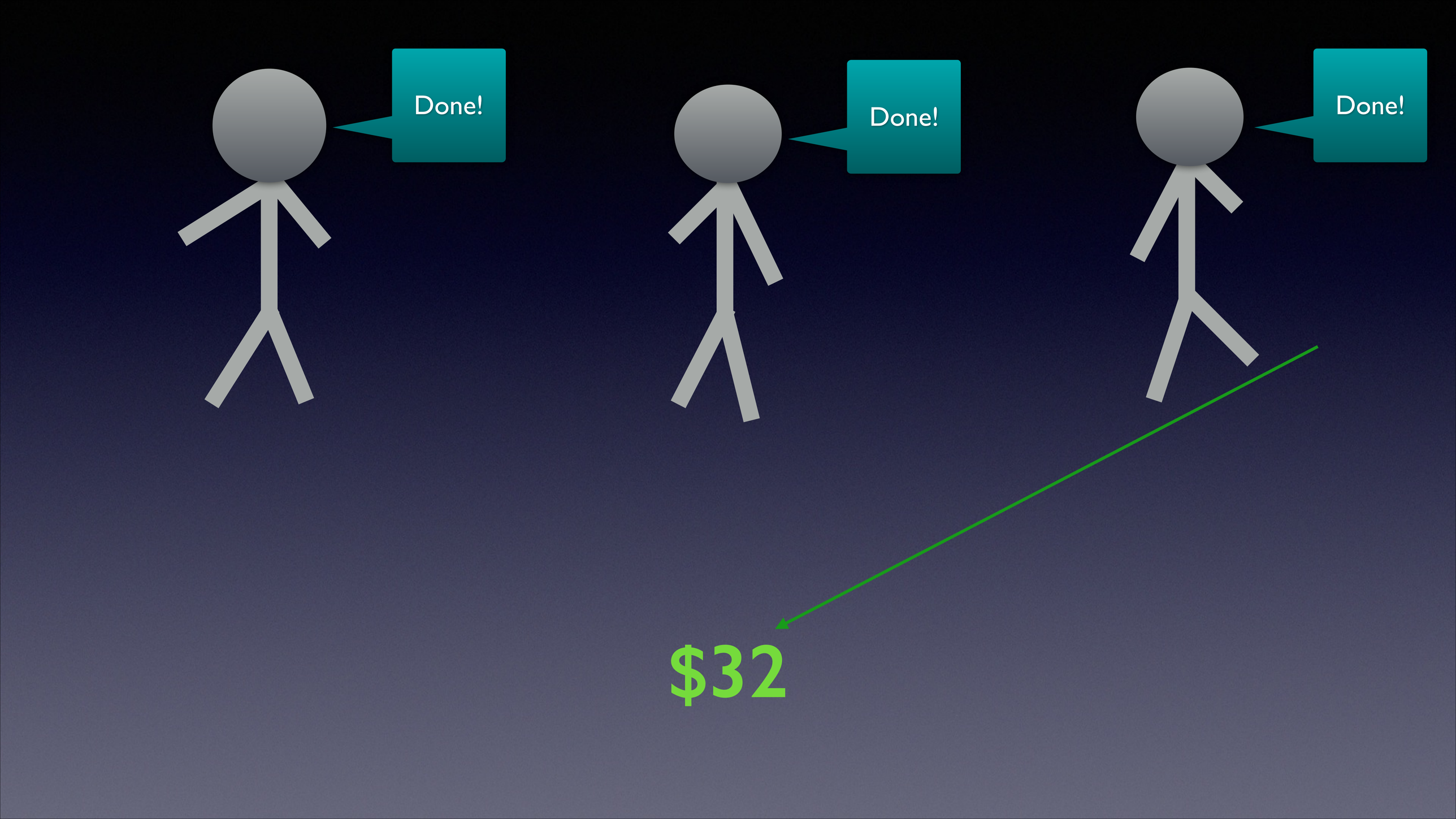


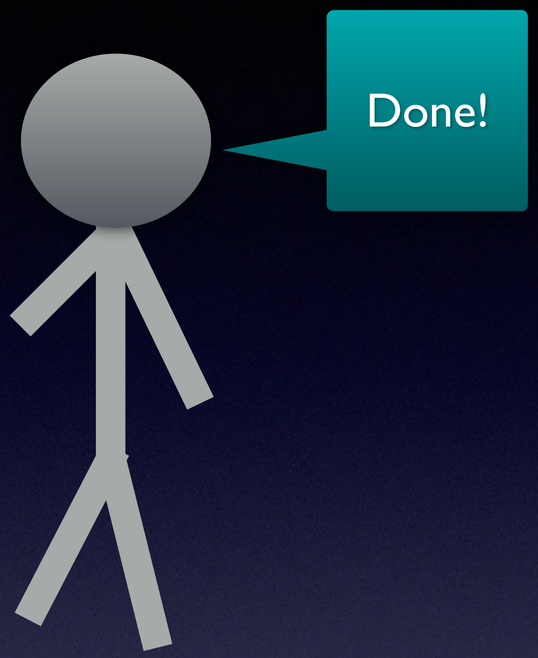
$9

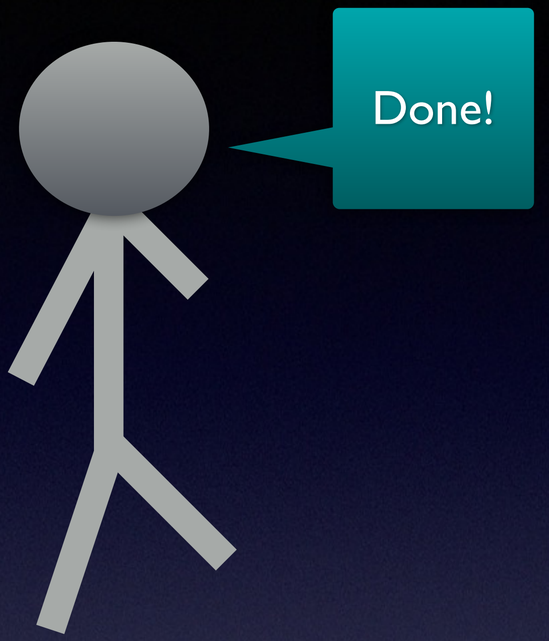

$23
Done!
$18
$23 $0
$8 $10
$1


Done!
Done!
$18
$8 $10
$24


Done!
Done!
$18
$18
$6


Done!
Done!
Done!
$32





$32

Как это использовать?

- Допустим, ресурс предоставлен процессу
- В каком состоянии система?
  - Если в безопасном – предоставить ресурс
  - Если в небезопасном – заблокировать процесс

Анализ этого подхода

+ не нужно “отбирать” ресурсы

+ меньше ограничений по сравнению с предотвращением взаимной блокировки

- максимальное количество необходимых ресурсов должно быть известно заранее

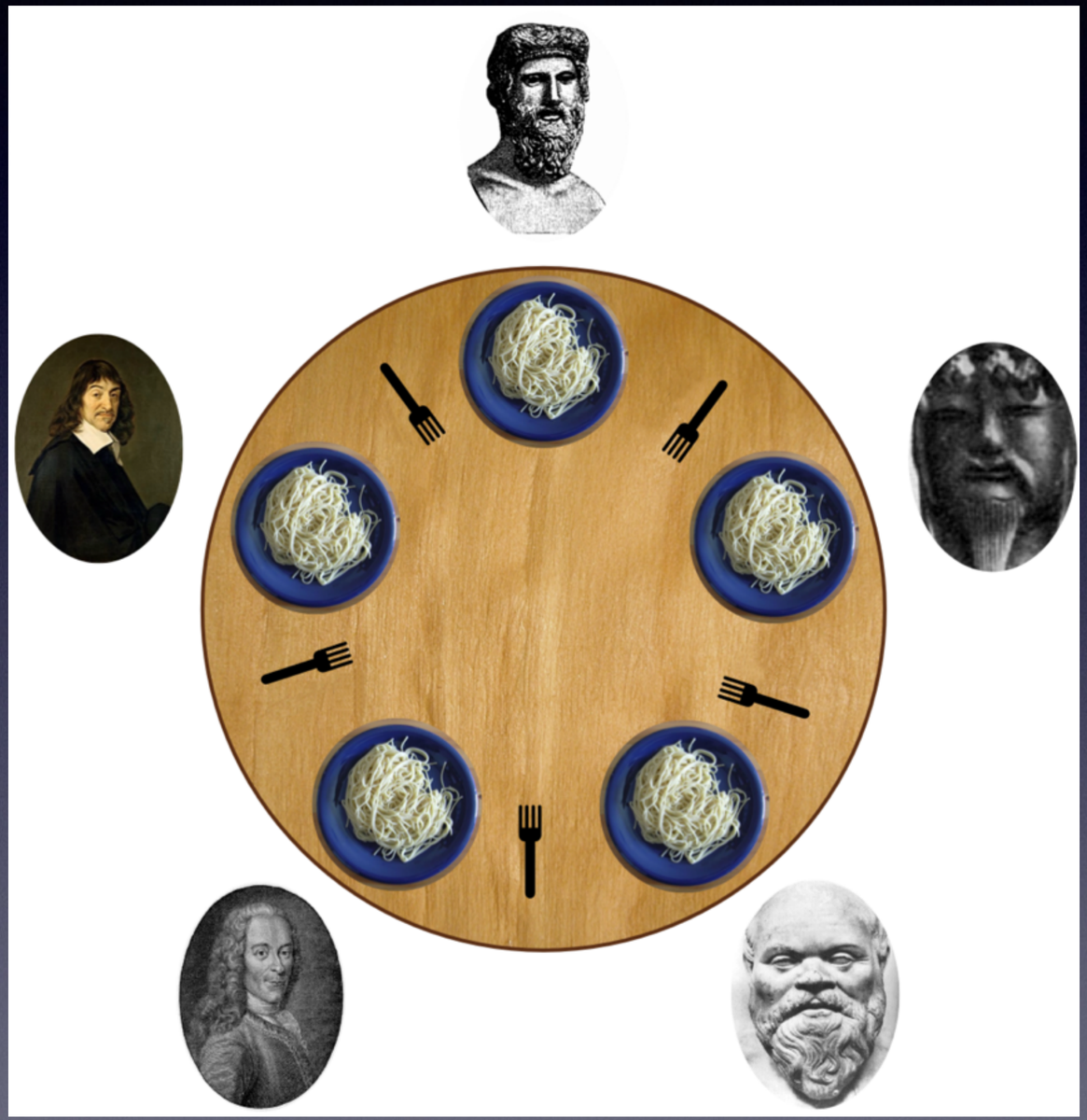
- количество ресурса должно быть конечным

- процесс не может завершиться пока он владеет ресурсом

Обнаружение

- В противоположенность предотвращению – можно все, но всегда нужно следить за состоянием

Проблема обедающих философов



Проблемы

- Взаимная блокировка
- Ресурсное голодание

Решения

- Официант (семафор)
  - философы спрашивают разрешения
- Иерархия ресурсов